НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації і управління

«**До захисту допущено**»
В.о. завідувача кафедри
_____ __*О.А.Павлов*__
(підпис) (ініціали, прізвище)
“___” _____ 2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки *6.050103 «Програмна інженерія»*

спеціальність *«Програмне забезпечення систем»*

на тему:*Автоматизована централізована система управління бекапом баз даних*

Виконав: студент 4 курсу, групи ІП-52

Набоков Едуард Максимович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *доцент, ф-м.н., доцент Гавриленко О.В.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант з графічної документації *доц. к.т.н. Ліщук К.І.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент *доц. каф. ТК, д.т.н. Борукаєв З.К.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	КПІ.ІП-5212.045430.02.81	«Автоматизована централізована система управління бекапом баз даних».		
			Пояснювальна записка	75	
3	А4	КПІ.ІП-5212.045430.03.91	«Автоматизована централізована система управління бекапом баз даних».		
			Технічне завдання	12	
4	А4	КПІ.ІП-5212.045430.02.81	«Автоматизована централізована система управління бекапом баз даних».		
			Програма та методика тестування	14	

АНОТАЦІЯ

Робота включає у себе 75 сторінок, 11 рисунків, 26 таблиць та 25 джерел.

До бакалаврської дипломної роботи Набокова Едуарда Максимовича на тему: «Автоматизована централізована система управління бекапом баз даних».

Мета дипломної роботи полягає у розробці централізованої системи, яка здатна автоматично створювати бекапи за заданим часом або у разі негайної потреби – екстреного створення бекапу та відновлювати їх у базі даних через універсальний інтерфейс кожної з баз даних.

Об’єктом дослідження є пошук підходів до створення універсального інтерфейсу, який дозволяє взаємодіяти з будь-якою базою даних.

Під час виконання даної роботи було проаналізовано методи створення бекапів баз даних, їх концептуальна різниця. Було розглянуто альтернативи до механізму спілкування мікросервісів – REST. Було запропоновано та доведено коректне використання підходу - RPC. Було виведено універсальний інтерфейс для взаємодії з базами даних. У рамках роботи було реалізовано мікросервісну архітектуру на мові Golang з використанням gRPC та Protobuf, яка інтегрується у будь-яку існуючу інфраструктуру у різних хмарних платформах.

Результатом роботи – автоматизована система, яка надає централізований доступ для управління – створення та відновлення – бекапами різних баз даних використовуючи при цьому єдиний та універсальний інтерфейс для взаємодії з ними на базі RPC механізму. Систему легко інтегрувати в існуючу інфраструктуру на різних хмарних платформах.

КЛЮЧОВІ СЛОВА: RPC, ЦЕНТРАЛІЗОВАНІ СИСТЕМИ, РЕЗЕРВНІ КОПІЇ БАЗ ДАНИХ, МІКРОСЕРВІСНА АРХІТЕКТУРА, ХМАРНИЙ АГНОСТИК, МУЛЬТИПЛАТФОРМЕННІСТЬ.

ABSTRACT

The work includes 75 pages, 11 figures, 26 tables, 25 references.

Abstract to the bachelor thesis work of Eduard Nabokov: «Automated centralized system of management of databases backup».

The aim of the thesis is devoted to the development of the centralized system, that is capable to either automatically create backups by scheduled time or in case of urgent demand – instantly create backups and restore them into databases via the universal interface for each of databases.

The object of research is the searching of approaches for creating universal interface, which can be used to interact with any database.

During the implementation of the current work were analyzed the methods of creating databases backups, their conceptual difference in design. It was considered alternatives for mechanism of microservices' communication – REST. It was proposed and assured the correct use of the following approach – RPC. It was derived the universal interface for interaction with databases. There was implemented microservice architecture in Golang programming language using gRPC and protobuf within this work, which can be easily integrated into exisiting infrastructure on different cloud platforms.

The end game of this work is automated system, that provides centralized access for managing – creation, restoration – backups of various databases using common and universal interface for interaction with them based on RPC mechanism. System can be easily integrated into existing infrastructure on any cloud platforms.

KEYWORDS: RPC, CENTRALIZED SYSTEMS, DATABASE BACKUPS, MICROSERVICE ARCHITECTURE, CLOUD-AGNOSTIC, CROSS PLATFORM.

ВМЕСТО ЭТОГО ЛИСТА ВСТАВИТЬ ЛИСТ ТИТУЛА ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Бекап – резервна копія бази даних.

CRUD – Create, Read, Update, Delete – головні операції для взаємодії з базами даних.

REST - REpresentational State Transfer - механізм спілкування сервісів у мережі або локально. Популярний підхід для реалізації спілкування мікросервісної архітектури завдяки використанню API у вигляді Context Path. Застосовується для виконання CRUD операцій та передачі станів між сервісами.

RPC - Remote Procedure Call – альтернативний підхід до REST. Націлений на виконання команди на віддалених серверах через певний інтерфейс.

GRPC – реалізація RPC протоколу мовою Golang. Проект заснований компанією Google.

Protobuf – Protocol Buffers – протокол для серіалізації даних під час передачі між сервісами.

GOROUTINES – програмні потоки, які керується runtime-ом мови Golang.

GOMAXPROCS – максимальна кількість створення операційних потоків мовою Golang.

HDFS - Hadoop Distributed File System – розподілена файлова система у Hadoop системі. Реалізована на Java.

HBase – нереляційна стовпчикова база даних від Apache. Існує в екосистемі Hadoop для зручного керування файлами. Реалізована на Java. Працює поверх HDFS.

Splice – метод проксіювання даних через операційний ріре не виділяючи при цьому короткочасну память для даних у user space.

Cron job – процес, який виконується у фоні. Не блокує основний процес виконання програмного забезпечення.

ВСТУП

Останнім часом з’являються нові системи. Кожна виконує свою роботу та несе відповідальність за свою працю. Існує безліч систем. Для логування та моніторингу серверів, виявлення уразливих місць інфраструктури для проведення атаки, що несе за собою можливе проникнення в іншу систему. Такі системи зручні у використанні, здатні сповіщати команду у разі появи або спрогнозованих проблем та є обов’язковими у налагодженні в світі системного та девопс адміністрування. З появою хмарних платформ – усі high-load системи розгортаються у Amazon Web Services, Microsoft Azure, Google Cloud Platform, International Business Machines, Digital Ocean, Yandex [1]. Такі хмарні середовища вирішують багато сучасних проблем.

З часом інфраструктури стають громіздкими. Ймовірність помилки зростає, контролювати стає складніше. Потрібно багато зусиль та ресурсів. Підняття інфраструктури у різних регіонах Землі, забезпечення fault та partition tolerance, завантаження мікросервісної архітектури, безлімітне сховище даних. За все це несе відповідальність кожна з платформ, яка це підтримує. Сучасний світ - інформаційний. Практично кожен проект використовує базу даних. Зчитує, обробляє/трансформує та зберігає або перенаправляє інформацію. Вона може бути різною за значенням та вагою. Демографічна статистика, історія пошуку в браузері, банківські рахунки користувачів, GPS альманах. Для цього використовуються бази даних та для забезпечення надійності збереження даних – репліки або резервні копії [2].

На цей рік кількість баз данних нараховує більше 30 популярних [3] та 100 неофіційних. Кожна має свої інструменти для управління. Бази данних поділяються за концепцією, масштабуванням, варіантами зберігання данних. SQL та NoSQL. Стовпчикова, документна, графова та ключ-значення [4]. Кожна вирішує певні задачі.

З появою нових баз даних складно інтегрувати нову БД, так як

виникає проблема у автоматизації управління БД. Це потребує значних змін у скриптах автоматизації, що призводить до втрати часу. Даний проект вирішує головну проблему з великою кількістю БД – пропонує автоматизований централізований єдиний універсальний інтерфейс для взаємодії з backup/restore будь-якої бази даних по всій інфраструктурі завдяки реалізованим плагінів/модулям до кожної з баз даних.

.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Автоматизована централізована система – система, яка дозволяє керувати централізовано процесами у різних місцях (сервери). Найпопулярнішими такими системами вважаються централізовані системи моніторингу та логування. З назви походить їхня сутність. Такі системи дозволяються контролювати сотні серверів водночас по всій інфраструктурі. Це полегшує роботу системних адміністраторів та дозволяє їм не контролювати це самостійно. Більше того, такі системи підтримують сповіщення. У разі виникнення проблеми – системний адміністратор дізнається про це завдяки сповіщення. Особливо це помітно у ночі. Проблеми на серверах можуть бути різними і залежить від прямої задачі системи. У випадку моніторингу – контроль виділеної оперативної памяті для процесів, виділених ресурсів для оперування даними, пропускної спроможності мережі або кількість IO операцій диску. У випадку логування – збір записів програм та зручна агрегація та композиція цих записів централізовано користувачу.

Розрізняють також Push та Pull моделі. Сервер може слухати на деякому порту та отримувати записи або метрики по TCP протоколу – push модель. Або ж сервер може обходити кожен з агентів на сервері та запитувати у нього про нові метрики. Обидві моделі вважаються рівносильними з невеликими нюансами. Не зважаючи на це, проблеми зявляються у разі некоректного доступу агентів до центрального серверу або навпаки.

Архітектура централізованих систем повинна притримуватись певних правил задля довгої працездатності, стійкості до помилок, надійності, консистентності, ефективності. Правила наступні:

- надлишковість (англ. redundancy). Центральний сервер – єдина точка відказу (англ. Single Point of Failure). Така проблема вирішується створенням дублікату центрального серверу та підключення його у разі провалу першого;

- доступність (англ. accessability). Кожен агент повинен мати змогу доступитись до центрального серверу. Приватні мережі повинні мати спільну маску у разі доступу по приватому IP адресу. Інакше, по DNS запису;

- шифрування (англ. encryption). Задля надійності даних, їх цілістностні та дійсності потрібно забезпечити відправку даних через TLS або HTTPs протокол з підтримкою HMAC;

Для реалізація спілкування у мікросервісній архітектурі використовуються два популярних механізми – RPC та REST.

Сутність цих механізмів:

- REST – механізм, який використовується у HTTP та використовує XML, JSON як одиницю спілкування. Полягає у передачі стану іншим мікросервісам за допомогою шляхів, які можуть бути використаними іншими сервісами для взаємодії;
- RPC – ще один механізм для спілкування з іншими сервісами. Націлений на виконання команд на віддаленому сервері як протокол SSH (англ. Secure Shell). Порівняльна характеристика REST та RPC наведена у таблиці 1.1.

Таблиця 1.1 - Порівняльна характеристика REST та RPC

	REST	RPC
Поняття	Representational State Transfer	Remote Procedure Call
З чим працює	- JSON - XML	- Protobuf
Де використовується	- Гіпермедія (HTTP, HTTPs) - CRUD операції	Виконує команди відаллено
Які HTTP методи використовуються	- GET - HEAD - POST - OPTION - PUT - DELETE - TRACE - PATCH	- GET - POST

Продовження таблиці 1.1

Ключові елементи	Спілкування сервісів через реалізовані шляхи (англ. path). Виду: /api/v1/user1/.	Реалізація інтерфейсу через який спілкуються сервіси.

На сьогодні існують реалізації RPC, REST механізмів. Один із широко відомих – gRPC – реалізація на мові Golang від команії Google.

Golang – статично типізована мова, яка набирає популярності на IT-ринку. Використовує легкі програмні потоки. Керуються вбудованим runtime-ом мови всередині операційних потокі. Максимальна кількість операційних потоків задається глобальною змінною – GOMAXPROCS. Ці потоки називаються goroutines. Альтернатива до Python – coroutines. Але це різні речі. Coroutines виконуються в одному операційному потоці (event loop) задля найменошого перемикання контексту та уникнення проблем з Global Interpreter Lock. Кожна goroutine має свою чергу для утримання задач та їх виконання у разі звільнення ресурсів. При створенні нового потоку – він має можливість забрати з іншої черги goroutine для виконання. У разі довготривалої задачі (важкий алгоритм)– goroutine преривається та береться наступна.

Цей механізм дозволяє швидко параллелити запити, що впливає на продуктивність.

Важко уявити, проект без використання баз даних. Вони різняться своїми принципами та вирішенню певних проблем. Не існує таких баз даних, які вирішуються всі проблеми. Основні типи баз данних, їх переваги та недоліки розглянуто у таблиці 1.2.

Таблиця 1.2 - Типи баз даних та їх характеристики

Назва напрямку	Переваги	Недоліки
Реляційні	Характеристика ACID. Дозволяє виконувати запити на обновлення – атомарно. У разі провалу існує rollback. Транзакції виконуються асинхронно. Наявність блокування рядків. Приклади: Postgres, MySQL, MSSQL.	Важке масштабування, шардування. У разі нормалізованих таблиць та їх розповсюдження – зростає час на такі операції як JOIN, HAVING.
Нереляційні	Характеристика CAP. Дозволяє зберігати документи, ключ-значення, залежності у графових БД. Пошук по документах.	Оновлення даних у нереляційних БД здійснюється оптимістичним та песимістичним блокуванням. Песимістичне підтримуються не всі БД. Підтримує MongoDB.

ACID походить від Atomicity, Consistency, Isolation, Durability. Набір характеристик, якими керуються реляційні бази данних. Atomicity – відповідальне за атомарність операції – транзакції, яка складається із запитів. Consistncy – відповідальне за валідний стан бази данних після здійснення транзакції. Банківска система: під час сплати за послугу чи товар – знімаються кошти з одної картки та додаються до іншої картки. Працює у рамках однієї транзакції. Isolation – транзакції виконуються незалежно один від одного та

ассинхронно. Durability – відповідальне за те, що кожна транзакція, яка була закомічена – залишиться у базі данних навіть після некоректного завершення бази данних.

CAP походить від Consistency, Availability, Partition tolerant. Набір харакеристик, якими керується нереляційна база данних.Consistency – відповідальне за те, щоб синхронізувати дані між серверами якнайшвидше задля нових запитів. Availability – кластер бази данних повинен бути доступним у будь-який момент часу. Partition tolerant говорить про те, що після поломки частини кластеру – база данних повинна продовжувати працювати. Проблема цієї теореми у тому, що неможливо одночасно надати жорстку консистентність та повну доступність. Окрім цього, розділяють консистентність на такі типи як:

- сильна (англ.strict) – кожен з читачів (reader) – отримує найостанніше оновлення данних у будь-який момент часу;
- послідовна – надає змогу виконувати оновлення рядків послідовно. При цьому кожен новий процес зчитування, отримує той рядок, який було оновлено послідовно;
- слабка – зчитування у будь-який момент часу не гарантує найостаннішу версію даних.

Нереляційні діляться на сфери рішення задач. Тип нереляційних баз данних та які проблеми вона вирішує наведено у таблиці 1.3.

Таблиця 1.3 - Тип нереляційних баз даних та проблеми їх вирішення

Тип баз даних	Опис	Сфера рішення проблем або задач	Приклади
Документні	Зберігають ієрархічно-структурні документи.	Зберігання документів у певній ієрархічній структурі.	CouchDB, Couchbase, MongoDB, DynamoDB

Продовження таблиці 1.3

Ключ-значення	Хеш-таблиця, яка приймає на вхід два параметри: ключ та значення. Ключем може бути строка, число, бінарна послідовність. Пошук по ключу, Пошук по значенню. Можливе встановлення часу, коли ключ-значення вже не дійсні. Мають свій DSL для запитів.	Кешування або пошук сервісів, контейнерів (англ. service discovery) Зручно використовувати при новому завантаженню програм в існуючу інфраструктуру.	Consul, Redis, ElasticCache, Zookeeper, etcd.
Графові	Абстракція моделі у вигляді графу, яка дозволяю встановлювати звязки між об’єктами та суб’єктами. Пошук по найближчим сусідами.	Можлива побудова графу соціальної мережі, структури персоналу в компанії. Наочне представлення їх залежності.	Neo4J

Для опрацювання великих даних (англ. Big Data) існує такий механізм як розподілена файлова система (англ. Distributed File System). Її задача – розподілення даних на декілька серверів. Існують вже декілька реалізацій такого підходу, які наведені у таблиці 1.4.

Таблиця 1.4 - Реалізація розподілених файлових систем

Назва	Опис
HDFS	Розподілена файлова система у системі Hadoop. Оперує блоками у розмірі від 64MB – 512MB. Поєднує декілька незалежних серверів у єдиний кластер. Має NameNode як головний сервер, DataNode як другорядний для оперування даними на інших серверах.
Ceph	Сховище блоків, файлів, обьектів. Аналог HDFS.
Fuse	Модуль для операційних систем Linux. Дозволяє створювати файлові системи.
GlusterFS	Аналог HDFS, працює поверх ext4, xfs. Використовує fuse. Застосовують у хмарних платформах.
NFS	Протокол спілкування між серверами для поєднання файлових систем у єдину. Підтримує багато файлових систем.

Робота з такою файловою системою нічим не відрізняється від стандартної, крім додаткового інструмента та архітектури (namenode/datanode).

Сервіс namenode тримає у памяті мета дані про розташування блоків на різних серверах, де працює datanode сервіс.

Існують проблеми і з такими підходами. Для прикладу візьмемо найпопулярнішу розподілену файлову систему – HDFS. Її проблема полягає у тому, що вона може тільки додавати інформацію - блоки у розмірі 64 – 512MB - до файлів або видаляти їх повністю. Після виконання major/minor compation – малі HFile-и групуються у великі.

Для зручної праці з такими файлами було реалізовано стовпчикову базу даних – Hbase. База даних, яка дозволяє оперувати даними (файлами) швидко та змінювати їх зсередини. HBase має регіон сервер, який несе відповідальність за оперування та коректного опрацювання таких файлів.

Є декілька варіантів створення резервних копій HBase:

- використання написаної бібліотеки libhdfs на C, яка реалізує JNI інтерфейс для взаємодії з HDFS. Копіювання файлів з HDFS /hbase/data/, /hbase/archive;
- написання Java клієнта, який буде тригерити команди HBase-а;
- створення RPC клієнта на будь-якій мові, яка підтримує це. Підключення по порту 8020 до HBase та надсилання команд по RPC.

Практика показала, що простим способом вважається копіювання файлової системи, де зберігаються дані HBase (/hbase/data/). Проблема виникає при відновленні резервної копії назад у базу данних. HBase не побачить відновлені таблиці через HDFS. Перезавантажити регіон сервер або повністю HBase не допомагає.

Як було помічено раніше, усі дані в HBase проходять через регіон сервер та memstore. Тому такий варіант вважається простим у копіюванні, але складний у відновленні.

У разі написанні власного Java клієнта для взаємодії з HBase призводить до звуження простору використання іншими мовами програмування: Python, Golang, C/C++, Dart, Elixir. Окрім цього, для запускання такого клієнта

знадобиться Java Virtual Machine, яка займає певне місце в операційній системі. Більше того, це унеможливлює підтримку деякого проценту програмістів, які не володіються Java досконально.

Для забезпечення масштабності у пітримці з боку інших програмістів – краще використати RPC механізм. Це дозволить реалізувати різні варіанти створення резервних копій тих же баз даних на різних мовах програмування. При цьому універсальний інтерфейс та сама система не зміниться.

1.2 Змістовний опис і аналіз предметної області

1.2.1 Причини створення резервних копій

Статистика вказує на те, що у сучасному світі популярною помилкою втрати даних – людський фактор. Кожен робить помилки час від часу. Наприклад, збір CAN, GPS, TCPDUMP даних під час поїздок водіїв. На такий збір було витрачено кошти, зусилля та час водіїв і менеджерів. У разі випадкового видалення може призвести до катастрофічного положення стосунків між компанією, яка потребує ці дані та компанією, яка збирає ці дані. У разі запобігання такого випадку – необхідним є створення резервних копій, які дозволять відновити дані у разі їх втрати.

Наступною причиною – втрата даних під час передачі з одного сервера на інший. В сучасних інфраструктурах активну участь беруть message broker-и. Такі як RabbitMQ, Kafka, ZeroMQ, MQTT та подібні. Це звичайна черга, яка тримає у свої памяті повідомлення від одного сервера до іншого. У разі припинення роботи такої черги – втрата повідомлень може призупинити роботу сервісів, які очікують певного повідомлення. Користувач може проводити банківську операцію – і у разі втрати його транзакції, яка проходить через цю чергу – призведе до незадоволеного настрою користувача. Врешті, користувач припинить використовувати даний сервіс для банківських транзакцій. Правильним кроком зі сторони системних адміністраторів – це дублювання

таких меседж брокерів, які будуть існувати незалежно один від одного. Тобто створюємо надлишковість (англ. redundancy).

З появою шкідливих програм, які взламують, забруднюють та псують дані, які згодом не підлягають відновленню – створення резервних копій є необхідним та грають важливу місію у світі безпеки. Резервна копія повинна зберігатися у надійному та захищеному місці задля безпеки та у разі появи проблем з поточними даними, відновити до моменту цілих та не зіпсованих даних.

Ще одною причиною створення резервних копій – це можливість міграції з одного дата-центру в інший. Так як дата-центри незалежні одиниці всієї інфраструктури – міграція даних повинна відбуватися через третю сторону для забезпечення ізольованості центрів. Зважаючи на те, що резервна копія повинна зберігатись у сховищі – це сховище може відігравати роль третьої сторони.

1.2.2 Повне резервне копіювання баз даних, характеристики

Сутність такого підходу полягає в тому, щоб повністю скопіювати усі файли бази даних та зберегти копію в сховищі, яке незалежне від поточного місцезнаходження бази даних. У разі термінового припинення роботи серверу, де працює база даних – резервна копія повинна знаходитись окремо для майбутніх відновлень.

Повна копія – найкращий захист даних. У разі виникнення помилки в БД – ця копія дозволить реставрувати базу даних у новому середовищі і продовжить функціонувати та обслуговувати клієнтів. Це може викликати певний down time під час відновлення. Проте для користувачів – це буде прозоро та безболісно, так як усі їх дані цілі та невтрачені. Такий тип бекапу називають рівнем 0.

Попри те, що такий метод дуже простий та коректний, сучасний світ – інформаційний та великі інфраструктури володіють петабайтами даних. Для повної резервної копії такої кількості даних – потрібно багато часу – місяць або

декілька місяців. Інформація за цей час може втратити свою користь. Усі витрачені на це ресурси не виправдають таку резервну копію. Окрім цього, для прискорення створення копій – потрібно більше ресурсів, які будуть паралельно обробляти інформацію та зберігати її віддаленому сховищі.

Наприклад, у Hadoop системі існує сервісна одиниця – MapReduce [7]. Її задача – зчитати, обробити, зберегти [8]. ETL – базова концепція опрацювання даних в сфері «великих даних» [9]. MapReduce розбиває набір даних на частини – split, обробляє кожну з них та завантажує в інше сховище.

Переваги повного резервного копіювання баз даних:

- у разі стихійного лиха – забезпечує надійне відновлення усіх втрачених даних;
- уся резервна копія зберігається в одному файлі.

Недоліки повного резервного копіювання баз даних:

- у випадку великих даних – потребує вдвічі більше пам’яті для копій;
- потреба у великій кількості часу на створення повної копії;
- у разі нелегалізованого доступу до копії з третьої сторони – спричинює витік усіх даних.

1.2.3 Покрокове резервне копіювання баз даних, характеристики

Ідея такого підходу полягає у збереженні різниці між поточним повною копією та самою базою даних. Для коректного використання такого механізму потрібно створити для початку повну резервну копію. Після цього створювати покрокові копії. У разі втрати одного з проміжних копій відновлення повної бази даних неможливе [11].

Переваги покрокового резервного копіювання баз даних:

- швидкість;
- потребує виділення памяті в розмірі одного покрокової копії;
- може бути запущений декілька разів підряд. Кожна покрокова копія – пункт відліку для майбутньої копії.

Недоліки покрокового резервного копіювання баз даних:

- повільне відновлення повної бази даних, адже потрібно пройтись по усьому ланцюгу копій, які збираються в єдину повну базу даних;
- увесь ланцюг з копій повинен бути присутнім або доступним при повному відновленні. Інакше повне відновлення – неможливе.

1.2.4 Транзакційне резервне копіювання баз даних, характеристики

Суть такого підходу полягає в тому, щоб зберігати не дані, а операції над даними Тобто перехоплювати будь-які зміни у базі даних та накопичувати у сховищі. Змінами можуть бути будь-які операції CRUD (create, read, update, delete). Такий підхід дозволить швидко відновити базу даних до будь-якого моменту часу.

Переваги транзакційного резервного копіювання баз даних:

- швидкість;
- копіювання тільки журнал змін;

Недоліки транзакційного резервного копіювання баз даних:

- втрата даних, які були незакомічені у журналі логування.

1.2.5 Диференціальне резервне копіювання баз даних, характеристики

Такий метод полягає у тому, щоб копіювати усі дані, які були змінені з часу останньої повної резервної копії. Щоденна копія та повна копія під кінець тижня.

Переваги диференціального резервного копіювання баз даних:

- швидкість;
- попередні копії, крім повної, можна видаляти. Адже кожна наступна копія містить у собі усі попередні, окрім повної.

Недоліки диференціального резервного копіювання баз даних:

- розмір копії з кожним часом збільшується;
- потребує повну копію для коректної роботи;

- у разі втрати повної копії – відновлення бази даних – неможливе;
- у разі втрати проміжної копії – повне відновлення – неможливе.

1.3 Аналіз успішних ІТ-проектів

1.3.1 Аналіз відомих технічних рішень

Відомо три технічних рішення для створення резервних копій баз даних:

- логічне;
- фізичне;
- створення знімків (англ. snapshot).

Логічне створення резервних копій – це створення запитів до баз даних, які отримують структуру таблиць, схем та самі дані. Існує інструмент у кожної бази даних, яка підтримує даний механізм. Цей інструмент створює скрипти з Insert запитами до нової бази даних. Таким чином, після виконання такого скрипту – створюється резервная копія в іншій базі даних. По факту, достатньо зберегти такий скрипт, який суміщає у себе створення структури таблиць та схем і додавання даних. Такий підхід повільний, адже потребує у форматуванні з фізичних даних у логічні. Ще одним з недоліків є те, що скрипти не містять у собі конфігурційні файли, що призведе до некоректної роботи нової бази даних. Але перевага є в тому, що такий формат – платформо-незалежний, що робить можливим зберегти його як звичайний файл. Як правило, скрипти такого бекапу – дуже громіздкі, тому потрібно багато часу на створення такого скрипту та відновленюю бази даних, адже запити виконуються послідновно. Такий механізм дозволяє лише виконувати копії в real-time, поки база даних працює. Інакше, не зможе зробити запити до непрацюючої бази даних.

Фізичне створення резервних копій – це звичайна копія частини файлової системи, де зберігаються дані бази даних. Цей механізм набагато швидший за логічний: відсутнє форматування. Більше того, такий механізм копіює конфігураційні файли та логи бази даних. На відміну від логічного копіювання – фізичний нездатний скопіювати ті дані, що знаходяться в оперативній памяті,

так як вони не знаходяться ще на диску. Основна перевага цього методу – простота у тому, що достатньо скопіювати дані з файлової системи за допомогою стандартних інструментів операційної системи. Один із найголовніших недоліків – те, що копія, яка була зроблена на одному сервері з однією архітектурою не спроможна скопіювати на іншу архітектуру.

Створення знімків (snapshot) – це ще один із варіантів створення резерних копій, що полягає у тому, щоб копіювати за логічним принципом.

Розрізняють ще методи створення резерних копій:

- коли база даних вимкнена - offline;
- коли база даних ввімкнена, але ніхто не здатен обновлювати дані, тільки зчитувати дані - semioffline;
- коли база даних ввімкнена та повноцінно функціонує - online.

У разі вимкненої бази даних, відсутній доступ для запису нових даних. Достатньо скопіювати файлову систему, де збережено дані бази даних. Такі операції проводять увечері задля створення цілих копій без втрати інформації.

У разі бази даних в режимі «тільки зчитування» - створення копії може бути таким же ефективним, як і створення під час вимкненої бази даних.

У разі ввімкненої бази даних існує ймовірність того, що під час створення резерної копії, будуть дані, які були обновлені під час копіювання. Призведе до неконсистентних даних. Коли процес почне відновлювати дані – він впаде з помилкою про зіпсованість даних, або інший формат даних. Тому таку резервну копію повинні реалізовувати бази даних самостійно.

1.3.2 Аналіз відомих програмних продуктів

Сьогодні існує вже декілька успішних конкурентів, які реалізували один із механізмів копіювання баз даних. Кожен з конкурентів намагається реалізувати real-time копіювання. Адже такий підхід зможе запобігти відключення бази даних. Список успішних проектів та їх характеристики наведено у таблиці 1.5.

Таблиця 1.5 - Успішні проекти та їх характеристики

Назва	Опис	Ціна	Бази даних	Приватний чи публічний
IPerious	Націлений тільки на реляційні бази даних. Не підтримується. Працює тільки на операційній системі Windows.	150 евро	Oracle, SQLServer, MySQL, PostgreSQL, MariaDB.	Приватний
Pgbackman	Реалізація на мові Python. Копіює тільки Postgres. Централізований. Мануальні та заплановані резервні копії. Повідомлення та інформаційні джерела про резервну копію. Опрацювання виникнення помилок.	Безкоштовний	PostgreSQL	Приватний
HandyBackup	Займається копіюванням тільки реляційних баз данних. Активно підтримується та розвивається.	250 евро	MySQL, PostgreSQL, MariaDB, MSSQL, Oracle, IBM DB2.	Приватний

Продовження таблиці 1.5

Amanda	Підтримується і розвивається. Працює тільки на Linux.	Безкоштовний	MySQL, PostgreSQL, MSSQL.	Публічний
Urbackup	Займається копіювання файлової системи, а не баз даних. Широко розповсюджений. Проблема виникне при відновленні цих копій. У разі звичної файлової системи – чудовий інструмент. У разі баз даних – безкористний. Підтримує повні та інкрементальні копії. Крос-платформенні клієнти. Підтримка real-time копій. Працює тільки з NTFS форматом.	Безкоштовний	Не копіює бази даних. Тільки файлові системи з NTFS форматом.	Приватний
AWS Backup	Підтримується Amazon Web Services. Займається копіюванням тільки AWS продуктів. Працює в AWS.	Безкоштовний інструмент, але плата за кількість даних, яка зберігається у сховищі.	EBS, RDS, DynamoDB, EFS, Storage Gateway snapshot.	Приватний

1.4 Аналіз вимог до програмного забезпечення

Система працює тільки при наявності усіх перелічених сервісів. Необхідною умовою для існування та функціонування живої системи потрібно два сервіса: master та minion. Достатньою умовою для повноцінного функціонування та керування системою потрібно три сервіси: master, minion, dashboard.

1.4.1 Розроблення функціональних вимог

Таблиця 1.6 - Варіант використання UC001

Назва	Миттєве створення резервної копії бази даних
Опис	Користувач натискає на клавішу і одразу створюється резервна копія бази даних
Учасник	Користувач - системний адміністратор, девопс
Передумови	Відкрита голова сторінка, існування одного чи декілької підключених minion-ів
Постумови	Користувач створив бекап та у заданому віддаленому сховищу вже існує поточний бекап.
Основний сценарій	1) Користувач відкриває головну сторінку у веб-браузері сервісу; 2) Обирає одного з minion; 3) Обирає певну базу даних для управління її резервними копіями; 4) Натискає “Instant” клавішу для створення миттєвого бекапу; 5) Користувачу повертає відповідь про статус бекапу.
Розширення сценарію	Сервіс сповіщає користувача у разі помилки.

Таблиця 1.7 - Варіант використання UC002

Назва	Заплановане створення резервної копії бази даних
Опис	Користувач задає період створення резервної копії бази даних
Учасник	Користувач - системний адміністратор, девопс
Передумови	Відкрита голова сторінка, існування одного чи декілької підключених minion-ів
Постумови	Користувач запланував створення резервної копії
Основний сценарій	1) Користувач відкриває головну сторінку у веб-браузері сервісу; 2) Обирає одного з minion; 3) Обирає певну базу даних для управління її резервними копіями; 4) Задає годину о котрій повинен тригернутись minion для створення бекапу (щоденний); 5) Користувачу повертає відповідь про статус запланованого бекапу.
Розширення сценаріїв	Сервіс сповіщає користувача у разі помилки.

Таблиця 1.8 - Варіант використання UC003

Назва	Авторизування користувача через BasicAuth
Опис	Користувач входить в систему завдяки авторизації через логін та пароль
Учасник	Користувач - системний адміністратор, девопс
Передумови	Запущені dashboard та master сервіси
Постумови	Вхід у систему дозволений, користувач має можливість взаємодіяти з бекапами;

Продовження таблиці 1.8

Основний сценарій	1) Користувач відкриває головну сторінку сервісу; 2) Вводить поля логіну та паролю для автомаризації через BasicAuth; 3) Перевірка введених даних; 4) У разі коректних даних - вхід в систему
Розширення сценаріїв	Виведе помилку про невалідне авторизування

Таблиця 1.9 - Варіант використання UC004

Назва	Авторизування користувача через LDAP
Опис	Користувач входить в систему завдяки авторизації через LDAP creds
Учасник	Користувач - системний адміністратор, девопс
Передумови	Запущені dashboard та master сервіси
Постумови	Вхід у систему дозволений, користувач має можливість взаємодіяти з бекапами;
Основний сценарій	1) Користувач відкриває головну сторінку сервісу; 2) Вводить поля логіну та паролю для автомаризації через LDAP; 3) Перевірка введених даних; 4) У разі коректних даних - вхід в систему
Розширення сценаріїв	Виведе помилку про невалідне авторизування

Таблиця 1.10 - Варіант використання UC005

Назва	Відновлення певного бекапу у базу даних
Опис	Користувач може відновити повний бекап у поточну базу даних
Учасник	Користувач - системний адміністратор, девопс
Передумови	Існуючий повний бекап у віддаленому сховищі для обраної бази даних
Постумови	Повне відновлення бази даних з обраного бекапу
Основний сценарій	1) Користувач обрає один з існуючих бекапів для поточної бази даних; 2) Натискає на клавішу “Restore”.
Розширення сценарію	У разі неправильної конфігурації сервісу Minion – у логах з’являться помилки про підключення

Таблиця 1.11 – Варіант використання UC006

Назва	Додання нового minion як агента
Опис	Користувач встановлює агента на новий сервер
Учасник	Користувач - системний адміністратор, девопс
Передумови	Скомпільований сервіс під потрібний сервер, платформу, де буде запускатися агент
Постумови	На головній сторінці відобразиться мета інформація про поточного агента

Продовження таблиці 1.11

Основний сценарій	1) Користувач компілює вбудованими інструментами мови Golang сервіс отримуючи при цьому бінарний файл, який вже готовий до роботи; 2) Деплоїть цей бінарний файл засобами CI/CD або через scp (secure copy); 3) Задає у конфігураційному файлі IP адресу та порт master сервісу; 4) Відкриває головну сторінку – отримує метаінформацію про задеплоїний сервіс.
Розширення сценарію	У разі неправильної конфігурації сервісу Minion – у логах з’являться помилки про підключення до Master

Функціональні вимоги додатку описано наступними таблицями.

Таблиця 1.12 – Опис функціональної вимоги REQ001

Номер	REQ001
Назва	Авторизація у систему через BasicAuth
Опис	Сервіс авторизує користувача через Basic Auth

Таблиця 1.13 – Опис функціональної вимоги REQ002

Номер	REQ002
Назва	Авторизація у систему через LDAP
Опис	Сервіс авторизує користувача через LDAP

Таблиця 1.14 – Опис функціональної вимоги REQ003

Номер	REQ003
Назва	Миттєве створення резервної копії баз даних
Опис	Користувач має можливість у будь-який момент створити копію бази даних через єдиний інтерфейс.

Таблиця 1.15 – Опис функціональної вимоги REQ004

Номер	REQ004
Назва	Заплановане створення резервної копії баз даних
Опис	Користувач має можливість запланувати щоденне, щотижневе, щомісячне створення бекапу баз даних.

Таблиця 1.16 – Опис функціональної вимоги REQ005

Номер	REQ005
Назва	Додання нових агентів для управління бекапами певного сервера або кластеру
Опис	Користувач має можливість задеплоїти агента на бажаний сервер та керувати копіями баз даних віддалено через централізовний веб-інтерфейс.

Таблиця 1.17 – Опис функціональної вимоги REQ006

Номер	REQ006
Назва	Вказання на статус агенту через веб-інтерфейс
Опис	Користувач має можливість побачити поточний статус агенту на головній сторінці системи.

Таблиця 1.18 – Опис функціональної вимоги REQ007

Номер	REQ007
Назва	Перегляд простору імен, таблиць баз даних
Опис	Користувач може переглянути усі простори імен та таблиці поточної бази даних

Таблиця 1.19 – Опис функціональної вимоги REQ008

Номер	REQ008
Назва	Відновлення резервної копії у поточну базу даних
Опис	Користувач може відновити дані з резервної копії у поточну базу даних

Залежність між вимогами системи зображено у таблиці 1.20.

Таблиця 1.20 - Матриця залежності між вимогами застосунку і варіантами використання

	REQ001 Авторизація у систему через BasicAuth	REQ002 Авторизація у систему через LDAP	REQ003 Миттєве створення резервної копії баз даних	REQ004 Заплановане створення резервної копії баз даних	REQ005 Додання нових агентів для управління бекапами	REQ006 Вказання на статус агенту через веб-інтерфейс	REQ007 Перегляд простору імен, таблиць баз даних	REQ008 Відновлення резервної копії у поточну базу даних
UC001 Миттєве створення резервної копії бази даних			■	■			■	
UC002 Заплановане створення резервної копії бази даних			■	■	■		■	
UC003 Авторизування користувача через BasicAuth	■	■				■		
UC004 Авторизування користувача через LDAP	■	■					■	
UC005 Відновлення певного бекапу у базу даних			■	■	■		■	■
UC006 Додання нового minion як агента					■			

1.4.2 Розроблення нефункціональних вимог

Вимоги до системи:

- автоматизована;
- централізована;
- вміння керувати бекапами;
- кросплатформеність.

Веб-інтерфейс повинен бути десткопним.

Апаратні та програмні вимоги:

- три сервіси повині працювати на серверах;
- сервіс Minion повинен працювати на сервері з базами даних;
- сервіс Dashboard може працювати на будь-якому апартному пристрої. Потрібно правильно написати конфігураційний файл;
- сервіс Master повинен працювати на окремому сервері.

Операційні вимоги:

- відмовостійкість;
- надлишковість;
- централізоване управління;
- оптимізація плагінів.

1.4.3 Постановка комплексу завдань модулю

Призначенням розробки даного проекту – програмна реалізація системи. Вона автоматизована, централізована, вміє взаємодіяти з будь-якими базами даних через єдиний інтерфейс з допомогою плагінів до баз даних, контролю резервні копії.

Метою розробки – реалізація централізованої системи, яке керує резервними копіями баз даних. При цьому додаючи нову базу даних у будь-яку існуючу інфраструктуру – користувач автоматично отримує доступ до планування створення копій баз даних через централізований GUI.

Потрібно реалізувати наступні кроки для вирішення такої задачі:

- змоделювати мікросервісну архітектуру для зручного створення копій баз даних;
- вивести єдиний інтерфейс для взаємодії з базами даних через плагіни;
- реалізувати плагін для HBase за заданим інтерфейсом;
- створити веб-інтерфейс для взаємодії з міньйонами;
- реалізувати взаємодію між сервісами на RPC.

1.5 Висновки по розділу

Після проведення аналізу вимог програмного забезпечення було оцінено та досліджено предметне середовище. Було проаналізовано варіанти створення резервних копій. Оцінено їх швидкодію та виконуваність. Було звернено увагу на різницю між RPC та REST. Проаналізовано створення копій у базі даних – HBase. Виведено найкращий механізм – RPC.

Окрім цього, було проаналізовано успішні проекти – конкурентів. Було визнано переваги та недоліки кожного з них. Порівняно з іншими було виділено певні особливі та унікальні характеристи цього проекту.

Було проведено пошук альтернатив до HDFS. Оцінено їх реалізацію.

Були сформульвані концептуальні ідеї розробки програмного продукту: централізований доступ, автоматизована система, створення резерних копій через універсальний інтерфейс між базами даних, відновлення копій баз даних.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

2.1.1 Побудова системи створення копій баз даних та її аналіз

Розроблювана система повинна суміщати у себе ряд характеристик:

- автоматизація;
- централізованість;
- універсальний інтерфейс, який реалізовують плагіни до баз даних;
- створення резервних копії баз даних.

Автоматизацію можна розділити на процеси або сервіси, які будуть виконувати свою роботу і тільки одну функцію. Такий технічний підхід було виведено з реальних проаналізованих систем. У випадку одного процесу, який виконує багато задач одночасно, може привести до великих проблем із горизонтальним масштабування. Сучасний світ намагається виділити дискретні частини процесів та віднести кожен до певної задачі. Наприклад, у моніторинговій системі, яка зображена на Рисунку 2.1. Вона описує існування двох основних процеса:

- процес, який збирає метрики з сервера та відправляє їх до агрегуючого процесу;
- процес, який агрегує отримані метрики та аналізує їх.

Рисунок 2.1 - Схема взаємодії автоматизованих процесів моніторингової системи

Централізовані системи вирішують наступні задачі:

- контроль усіх створених нею процесів;
- контроль стану процесів;
- надання дистанційно задач процесам;
- контроль виконання задач;
- сповіщення користувачів у разі появи проблем.

Централізовані системи були запропоновані задля спрощення маніпуляції багатьма процесами. Активно використовуються у сучасних інфраструктурах системними адміністраторами. Так як такі системи полегшують їх роботу та потребують менше часу на діагностику з боку користувачів через єдину точку входу, Схематично централізовані системи зображені на Рисунку 2.2.

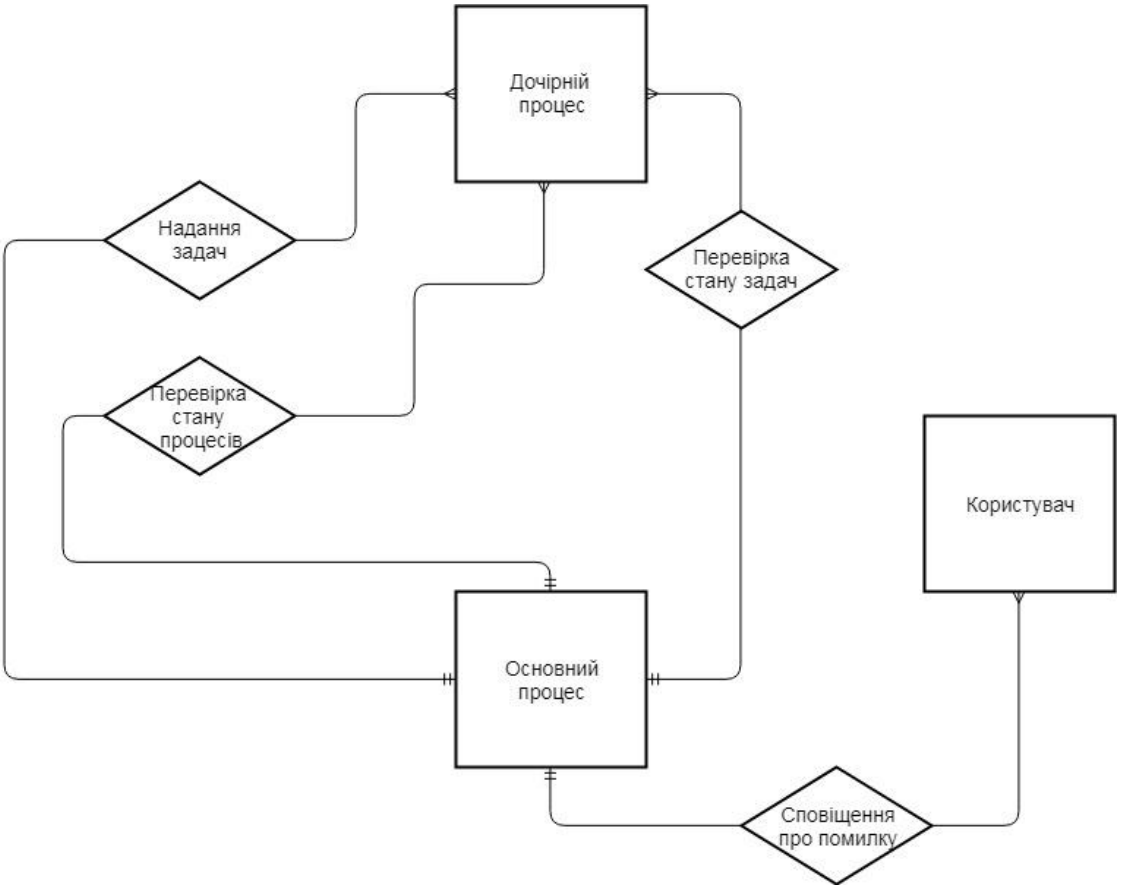

Рисунок 2.2 - Схема роботи централізованої системи

Для досягнення універсального підходу до кожної бази даних потрібно змоделювати універсальний інтерфейс. Кожна з нових баз даних повинна реалізовувати його - плагін. Це дозволить їй інтегруватись у розроблювану систему без зайвих виправлень у самій системі. Інтерфейс та плагіни до баз даних схематично зображено на Рисунку 2.3.

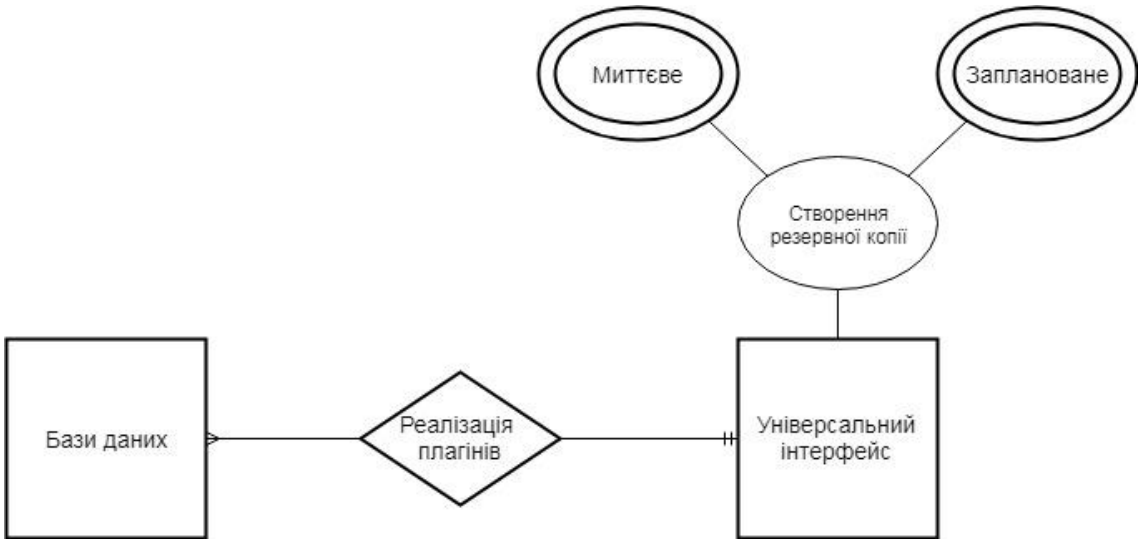

Рисунок 2.3 - Відношення плагінів до універсального інтерфейсу

Створення резервних копій – найголовніша частина моделювання системи. Розроблювана система повинна вміти використовувати плагіни задля створення копій певної бази даних. У даному проекті за основу було взято базу даних – HBase. Вже було розглянуто принципи створення резервних копій такої бази даних. Обраний метод створення копій – RPC. Схему створення резервної копії бази даних HBase зображено на Рисунку 2.4. RPC дозволяє використовувати різні мови програмування. Таким чином можно створити різні RPC клієнти незмінюючи при цьому сам механізм спілкування з HBase-ом.

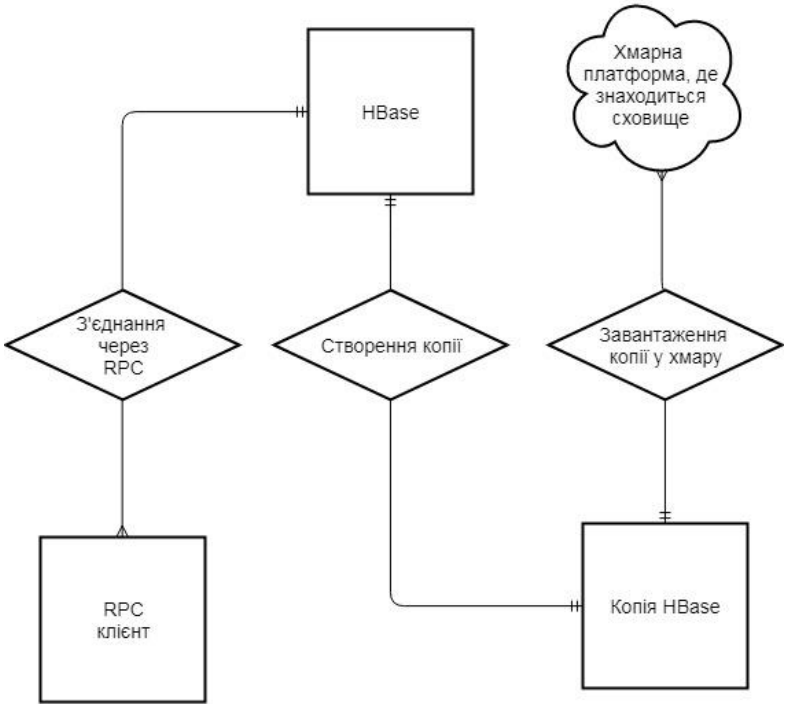

Рисунок 2.4 - Відношення об'єктів для створення копії баз даних

Схема взаємодії системи та користувача зображена на Рисунку 2.5

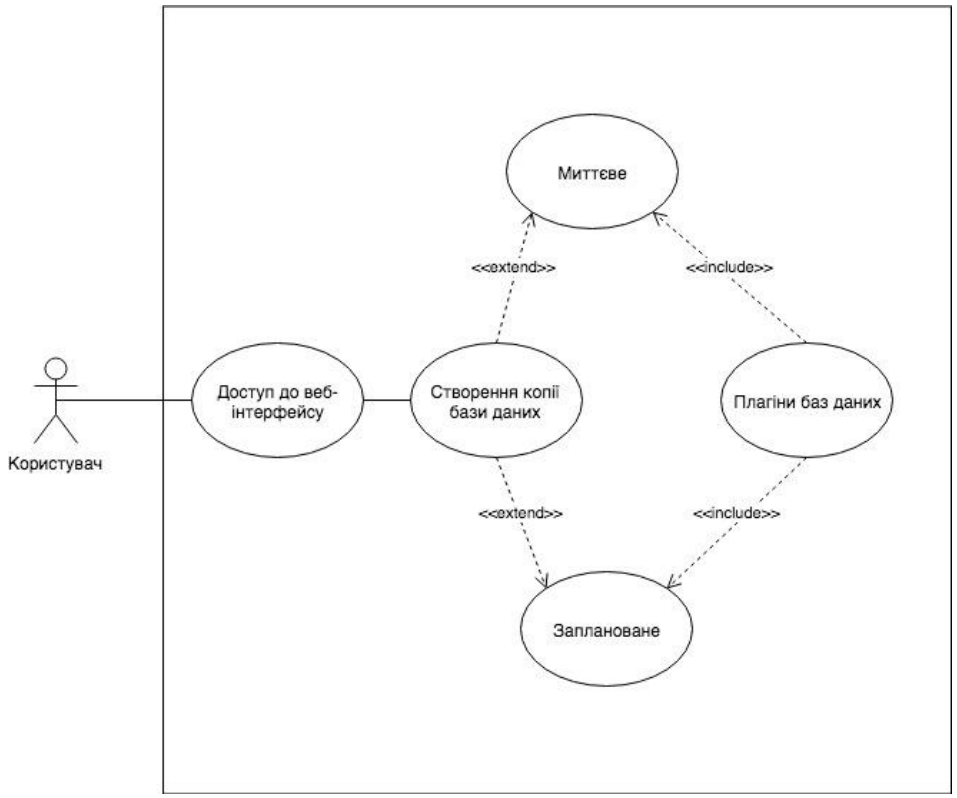

Рисунок 2.5 – Схема взаємодії користувача з системою

Послідовний опис такої схеми:

- користувач запрошує головну сторінку через веб-браузер;
- обирає будь-який з існуючих серверів для подальших операцій;
- натискає на базу даних для управління бекапами;
- створює миттєвий бекап або запланує його на заданий час;
- сервіс, який відповідає за створення бекапів, використовує плагін до бази даних через реалізований універсальний інтерфейс.

2.2 Архітектура програмного забезпечення

Діаграма класів програмного продукту наведена у графічному матеріалі. Детальний опис класів та функцій наведений у таблиці

Таблиця 2.1 – Класів, функцій системи та їх опис

Назва	Опис
_ConfMaster(Host, Port)	Структура, яка парсить частину конфігураційого файлу YAML статично, яка відповідає за задання IP адресу та порту сервісу Master.
_ConfS3Bucket(Name, Region)	Структура, яка парсить частину конфігураційого файлу YAML статично, яка відповідає за задання назви та регіону AWS S3 Bucket.
_ConfS3([]_ConfS3Bucket)	Структура, яка парсить частину конфігураційого файл YAML статично, яка відповідає за можливість задання декількох AWS S3 bucket-ів.

Продовження таблиці 2.1

_ConfNameNode(Host, Port)	Структура, яка парсить частину конфігураційого файл YAML статично, яка відповідає за задання IP адресу та порту HBase NameNode.
_ConfTarget(_ConfS3, …interface{})	Структура, яка парсить частину конфігураційого файл YAML статично, яка відповідає за задання будь-яких віддалених сховищ для збереження бекапу.
ConfMinion(Port, HeartBeat, Master, NameNode, Targets)	Структура, яка парсить частину конфігураційого файл YAML статично, яка відповідає за задання метаінформації для Minion сервісу.
ConfMaster(Port)	Структура, яка парсить частину конфігураційого файл YAML статично, яка відповідає за задання порту для Master сервісу.
Storage(Configs)	Структура, яка відповідає за збереження конфігураційних файлів.
Tasks	Структура, яка відповідає за збереження задач, які виконуються за заданим інтервалом.
ConnPool	Структура, яка управляє підключеннями, їх перевикористання та обмеження до певної кількісті паралельних підключень.

Продовження таблиці 2.1

Heartbeat	Функція, яка запускає cron job за заданим інтервалом в конфігураційному файлі, що пінгує Master сервіс для повідомлення про стан.
ScheduleBackup	Функція, яка запускає cron job за заданим інтервалом користувачем, що створює бекапи автоматично.
UnscheduleBackup	Припиняє роботу cron job.
GetPool	Повертає обьект для взаємодії з підключеннями RPC.
TableUnit	Структура, яка описує таблицю бази даних.
StrategyCLI	Структура, яка встановлює поточну базу даних та використовує її плагін.
GetAllMinions	Повертає метаінформацію про усіх агентів.
Master	Структура, яка являється центром сервісу Master.

Таблиця 2.2 – Методи класів та їх опис

Клас	Метод	Опис
IStorage	GetMasterConf	Повертає інформацію про master сервіс.
IStorage	GetMinionConf	Повертає інформацію про minion сервіс.

Продовження таблиці 2.2

IStorage	GetDashboardConf	Повертає інформацію про dashboard сервіс.
IStorage	Put	Зберігає заданий конфігураційний файл за певним ім’ям. Параметри: - filename – str – шлях до файлу; - injectName – str – назва конфігураційного файлу у сховищі; - out – interface – передання структури (описаних вище) для парсингу.

Продовження таблиці 2.2

Storage	GRPCConnect	Шукає вільні підключення до rpc Master та повертає їх. У разі не знайденого підключення – створює новий. Параметри: - ctx - передачі поточних глобальних змінних, які належать до певного підключення - host – string – задання IP адресу - port – int – задання порту
Storage	GRPCDisconnect	Зберігає поточне підключення для наступного перевикористання. Параметри: - conn - *grpc.ClientConn – обьект підключення.

Продовження таблиці 2.2

Database	GetNamespaces	Повертає усі простори імен поточної бази даних. Параметри: - socket – string – IP адреса та порт
Database	GetTables	Повертає усі таблиці поточної бази даних. Параметри: - socket – string – IP адреса та порт
Database	BackupSchedule	Заплановує бекап. Параметри: - socket – string – IP адреса та порт - namespace - string - tablename - string - timestamp - string - s3 – interface{}

Продовження таблиці 2.2

Database	BackupInstant	Створює миттєвий бекап. Параметри: - socket – string – IP адреса та порт - namespace - string - tablename - string - s3 – interface{}
Database	ListSnapshots	Повертає усі знімки поточної бази даних.
StrategyCLI	SetDatabase	Встановлює поточну базу даних Параметри: - cli – поточний обьект бази даних.
StrategyCLI	GetDatabase	Повертає поточну базу даних.
MetaClass	GetPrivateIP	Повертає приватний адрес поточного сервера.

2.2.1 Загальна схема

В архітектурі даного проекту будуть три сервіси, у яких інтерфейси реалізовані на gRPC та REST – master, minion, dashboard.

Планується створити сервіси, які є серверами та які можуть надавати запити до інших серверів. Проблема виникає у тому, що gRPC має клієнт-серверну архітектуру.

Повну схему взаємодії сервісів зображено на Рисунку 2.6.

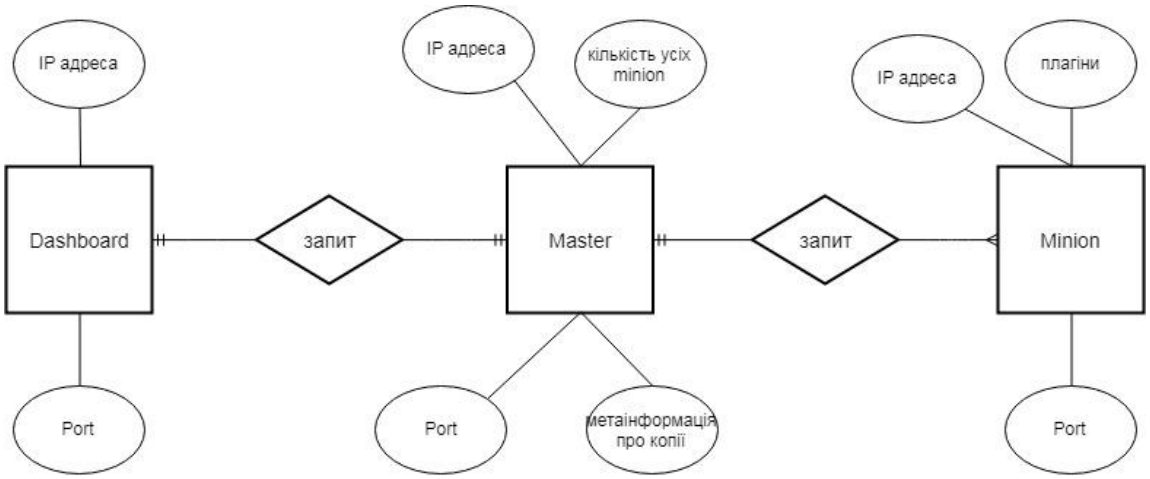

Рисунок 2.6 - Загальна схема взаємодії сервісів

Dashboard сервіс відіграє одну роль – GUI для взаємодії з Master. Має наступні характеристики та зобов’язання:

- представлення користувачу web-інтерфейс для взаємодії з Master сервісом;
- трансформація REST в RPC та навпаки;
- відображення простору імен таблиць, таблиці поточної бази даних;
- надання задачі Master-у.

Алгоритм виконання dashboard сервісу зображено на Рисунку 2.7.

Рисунок 2.7 - Алгоритм роботи сервісу dashboard

Наступний сервіс – Master. Слугує основним сервісом, який зберігає інформацію про

усі міньйони. Спілкується з ними через RPC. Надає їм команди для їх виконання. Контролює створення резервних копій обраних таблиць поточної бази даних. Алгоритм роботи Master сервісу зображено на рисунку 2.8.

Рисунок 2.8 - Алгоритм роботи сервісу master

Останнім з сервісів – Minion. Він встановлюється на серверах, де запущена база даних. При обрані певних плагінів, він під’єднується до бази та видає інформацію про неї. Для створення копій – сервіс master повинен тригернути цей сервіс. Основні задачі minion – створення резерних копій, вміти розмовляти з базами даних через реалізовані плагіни, сповіщати Master сервіс з певним інтервалом для перевірки здоров’я сервісу. У разі довгого відкликання – master помічає minion-а мертвим. Алгоритм роботи сервісу minion зображено на рисунку 2.9.

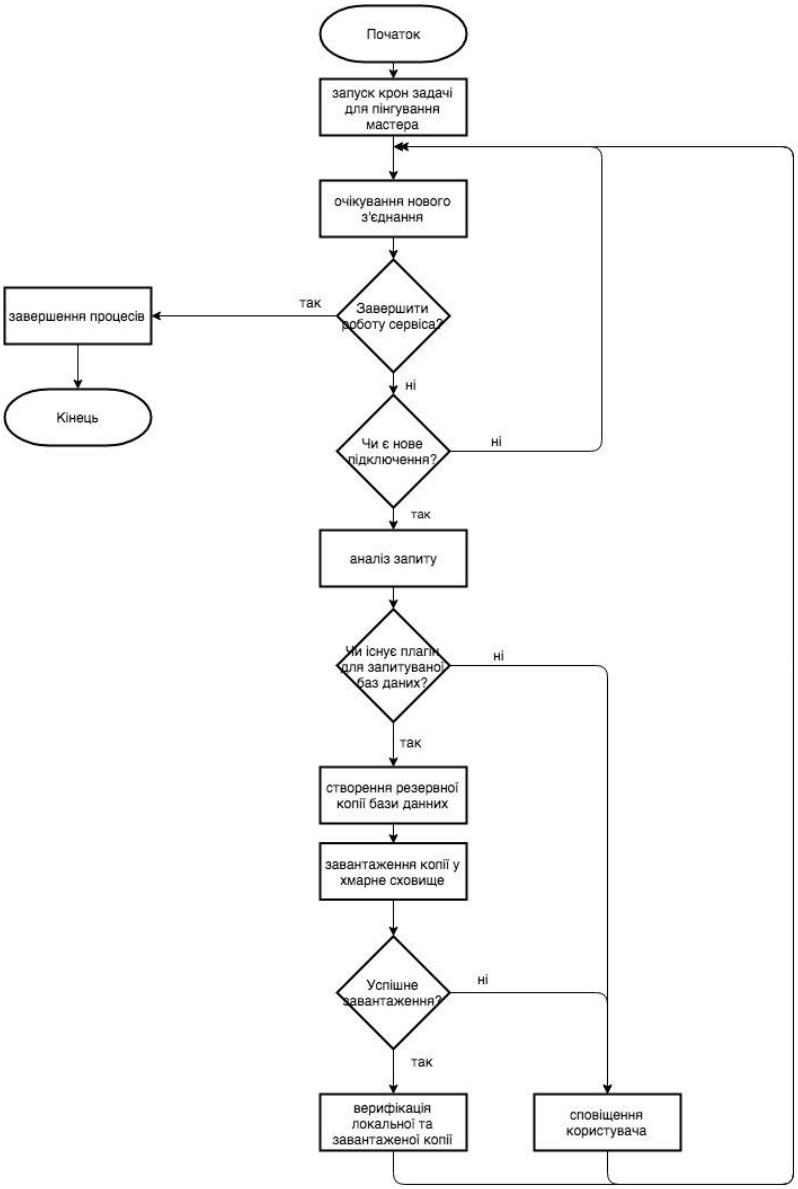

Рисунок 2.9 - Алгоритм роботи сервісу minion

2.2.2 Мова програмування для розробки мікросервісної архітектури

Для реалізації даного проекту потрібна мова програмування, яка задовільняє наступним характеристикам:

- статична типізація;
- швидке компілювання та виконання операцій;
- швидкий та вбудований збирач сміття (англ. garbage collector);
- мінімальна втрата памяті (англ. memory leak);
- наявність необхідних реалізацій клієнтів HBase, PostgreSQL, OracleDB та інші бази даних;
- підтримка паралельних обчислень (multithreading);
- підтримка RPC;
- підтримка REST;
- кросплатформеність.

Сьогодні існує безліч мов, які мають різні характеристики. Проаналізуємо найбільш популярні мови програмування як C/C++, Java, JavaScript, Golang.

2.2.2.1 C/C++

Розповсюджена мова програмування. На ній реалізовано безліч інших. Ця мова має статичну типізацію, компільована, поєднує як високорівневе програмування – HTTP сервіси, управління з файлами – так і низькорівневе – керування памятью, виділення памяті під об’єкти у heap space. Важко писати масштабовані програми. На написання таких програм потрібно багато часу. Практично реалізовувати багатопотокові програми. У разі з C/C++ - це може призвести до фатальних помилок, які концентрують увагу програміста не на реалізацію ідеї, а на виправлення цих помилок. У цієї мови вже є реалізації бібліотек, які підтримують основні протоколи включаючи HTTP (підтримка REST). Управління памятью – це складний процес, який потребує детальної діагностики задля запобігання втрат памяті. Незважаючи на популярність цієї

мови – в неї відсутній збирач сміття (garbage collector). Можна збирати проекти під різні архітектури та операційні системи.

Ця мова ідеально підходить під характеристики. Проте, вона потребує багато зусиль на написання даного проекту і переписування вже реалізованих бібліотек, але на інших мовах програмування.

2.2.2.2 Java

Статична і тільки високорівнева мова програмування. Тісно пов’язана з безпечним кодуванням. Контролює доступ до памяті. Має вбудований збирач сміття. Підтримує багатопоточність та асинхронність. Реалізовано безліч кастомних бібліотек для взаємодії з базами даних через JDBC драйвера. Підтримує RPC та REST.

Проблема цієї мови полягає в потребі JVM для підтримки кросплатформенності. У разі масштабування – кожен новий сервер потребує вже встановлений JVM для запуску мікросервісу на Java. Окрім цього, нещодавно компанія Oracle анонсувала те, що нові випуски Java будуть платними. Це вплинило негативно на більшість програмістів, які вже переходять на відкритий проект від Amazon Web Services – Corretto – дублікат останього релізу Java з розширеним новим функціоналом.

2.2.2.3 Javascript

Інтерпретована високорівнева мова програмування зі слабкою або динамічною типізацією. Має реалізації RPC, REST. Вбудоване та автоматизоване керування памятю. Існують реалізації клієнтів майже до більшості популярних баз даних. Не підтримує багатопоточність. Проте є реалізація асинхронного програмування з корутинами на event-loop. Зручна мова для написання веб-сервісів.

Ця мова не підходить за більшістю невідповідних характеристик.

2.2.2.4 Golang

Статично типізована мова програмування від компанії Google. Швидко набирає темп популярності. Сьогодні вже існує безліч реалізацій клієнтів до баз даних. Ця мова має вбудований garbage collector. Підтримує багатопоточність – goroutines, runtime, channels, queues. Має реалізацію RPC – gRPC. Підтримує REST. Кросплатформенний, адже будується під arm, i386, amd64 архітектури та операційні системи – Darwin, Android, Linux, Windows. Не потребує додаткових інструментів для запуску проекту. Достатньо вказани дві необхідні змінні середовища. З останнім релізом було переведено виділення памяті під операцію defer з Heap до Stack. Зараз мінімальна втрата памяті.

Для цього проекту Golang повністю підходить по характеристикам та на ньому легко реалізовувати свої ідеї в сфері IT.

2.2.3 Реалізація RPC для спілкування в мікросервісній архітектурі

Популярною реалізацією RPC є розробка від компанії Google – gRPC. Він має вбудовану мову для задання інтерфейсу. Вона жостко типізована. З цього інтерфейсу можна знегерувати різними мовами (C, Dart, Java, Node, PHP, Python, Go, Ruby) клієнтську частину, яка буде використана для спілкування. RPC, як і REST дозволяє спілкуватися сервісам незалежно від їх мови реалізації. Такий фремворк полегшує роботу взаємодії між самою системою та плагінами, які можуть бути реалізованими на будь-яких мовах.

2.2.4 Створення копій баз даних

У даному проекті було реалізовано роботу з HBase. Існує декілька варіантів створення резервних копій.

Найпростіший метод – просте експортування даних через вбудовані інструменти HBase-у на файлову систему HDFS. Цей метод викликає MapReduce задачу, яка у свою чергу викликає методи з HBase для експортування. Ця задача націлена на швидкодію. Великим нюансом такого

підходу – повне сканування даних та потреба у великій кількості памяті для збереження дублікату. Більше того, для завантаження потрібно реалізовувати свої методи Окрім цього, цей метод можна використати або через shell CLI або через Java клієнт.

Практика вказує на те, що такий метод повільний – адже спочатку він копіює дані на HDFS, і тільки потім завантажує їх до віддаленого сховища.

У такому випадку легким та швидким методом створення резервних копій – створення знімків бази даних. Знімок – snapshot, який суміщає у собі як мета інформацію так і самі дані, які зберігаються на HDFS та у оперативній памяті. Для пропуску зливання даних, які зберігаються у оперативній памяті можна використати спеціальну опцію, яке це пропускає.

Основна концепція роботи з HBase – RPC. Вона підтримується із коробки. Сгенерувавши stub для спілкування з HBase можна використовувати функції з програмного коду, як звичайний виклик функції. Основний функціонал може бути знайдений у офіційному репозиторію GitHub для HBase в директорії protocols.

У користувача є 2 варіанти створення резерних копій – миттєве та заплановане.

Потрібно мати на увазі, що створення копій потребує певних умов. У разі з HBase потрібно вимкнути таблицю для заборони подальшого запису. Щоб завантажити до AWS S3 потрібно мати пароль або приватний ключ. У випадку з роботою на AWS платформі – достатнью вказати IAM для цього інстансу. Більше того, кожне нове планування створення резерних копій – потребує ресурси.

Нехай існує одна база даних A та її резервні копії a. У разі існування ще однієї бази даних B – її бекапи – b. Тому загальною кількістю бекапів такої системи буде загальна множина усіх бекапів цих баз даних. Загальна формула копій зображено у формулі 1.1.

$$S = \{a1,\ a2,\ a3, \ldots,\ aj,\ b1,\ b2,\ b3 \ldots,\ bj\},\ i = 1, N,\ j = 1, M,\ ai \in A,\ bj \in B \qquad (1.1)$$

2.2.5 Завантаження копій у хмарні платформи

Для підтримки cloud-agnostic архітектуру – кожен з плагінів повинен сам реалізовувати завантаження копій. У цьому проекті для прикладу було обрано AWS S3 – широко відоме об’єктне сховище для збереження даних. Для копіювання між AWS S3 та HDFS було обрано механізм, який є стандартним для операційної системи Linux – splice. Механізм, який дозволяє копіювати дані між файловими дескрипторами або підключеннями, при цьому не виділяючи пам'ять для прохідних даних. Такий механізм забезпечує надійне завантаження у разі недостатньої памяті. Більш детатально про взаємодію процесів зображено на рисунку 2.10.

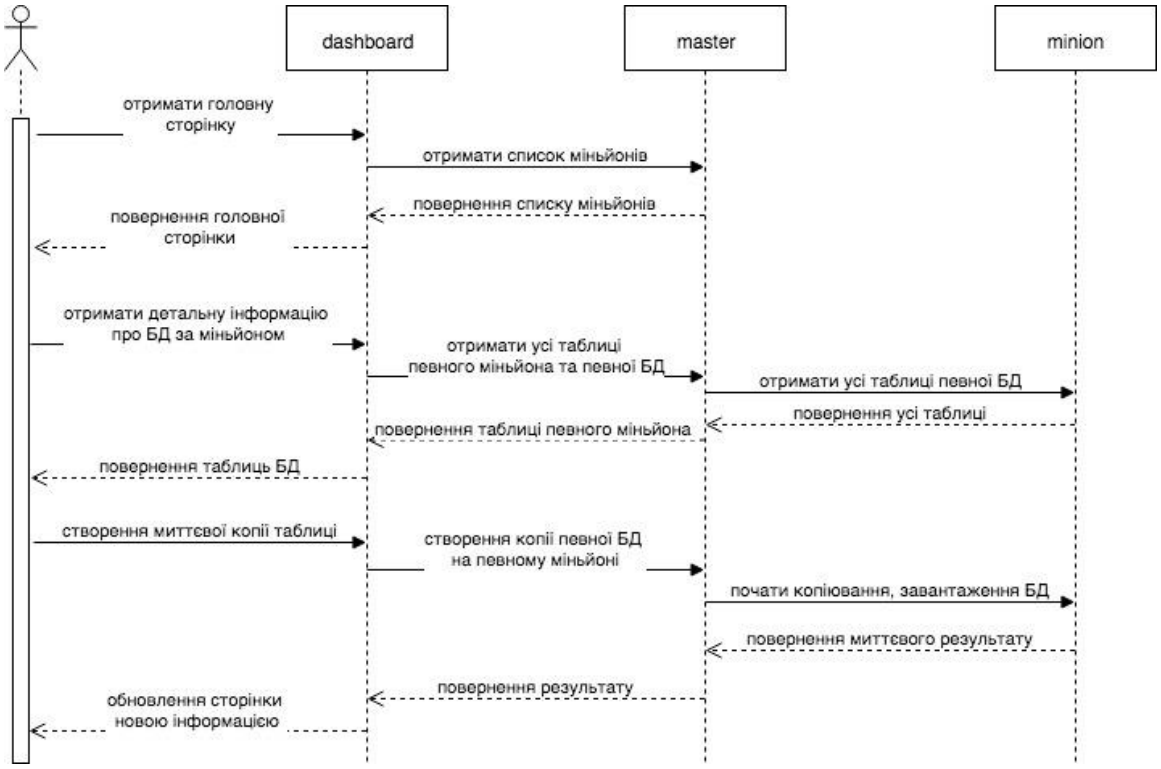

Рисунок 2.10 - Діаграма процесів між сервісами

2.2.6 Перевірка валідності скопійованих даних

Для валідації створених копій та їх завантежні копії використовується такий механізм як вирахування чексум – checksum – це хєш значення. Цей хєш береться з файлу. У разі зміни файлу – хєш буде іншим.

Такий принцип реалізовано у проекті для верифікації завантажених копії у платформу. Якщо хєшсуми локальної копії та завантаженої рівні – завантежння копії пройшло успішно. Інакше, виникли проблеми з передачею копії, третя сторона змінила зміст копії під час переді по TCP звязку.

Так як файл завантажується не однією великою частиною, а блоками – було б правильно вираховувати ще хешсуми кожного з блоків для більшої точності. Такий підхід використовується у разі синхронізації файлів між серверами або домашніми компьютерами. Популярним технічним продуктом такого підходу є Torrent, який для прискорення завантаження використовує інші сервера для покращення і пришвидшення розповсюдження даних.

2.3 Конструювання програмного забезпечення

Програмний продукт має два варіанти своєї роботи. Коли існує тільки Master та Minion. Відсутній Dashboard сервіс. В такому випадку можливе написання власного клієнта для спілкування з Master. Інший підхід – Master, Minion, Dashboard. Повноцінний функціонал для взаємодії.

Окрім цього, створено загальну бізнес модель. Бізнес схема повинна складатися з трьох основних процесів, які відповідальні за свою роботу. Процес оперування резервними копіями. Завантаження до хмарного середовища. Створення резервної копії баз даних. Бізнес модель та процесів наведено на рисунку 2.11.

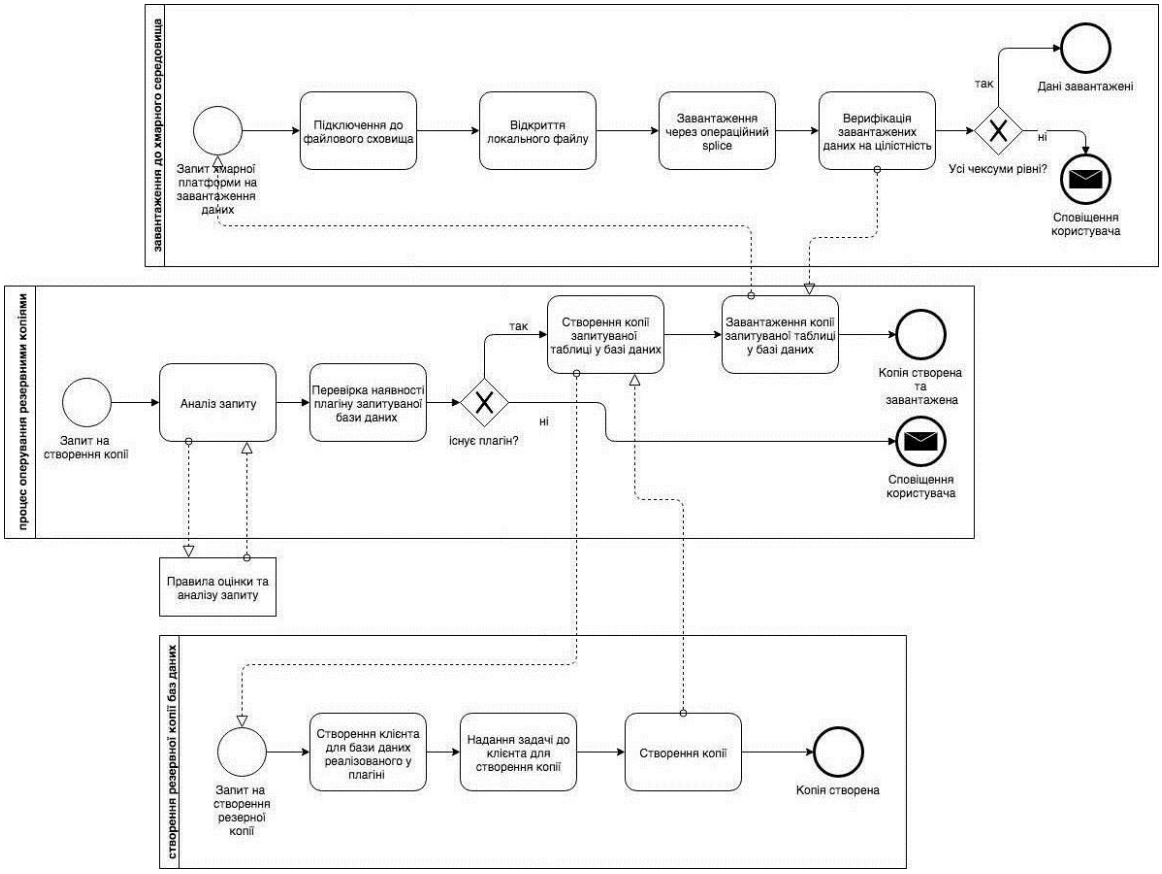

Рисунок 2.11 - Бізнес модель обробки запиту створення бекапу

Бізнес модель складається з трьох основних процесів:

- процес оперування резерними копіями;
- створення резервної копії баз даних;
- завантаження бекапу до хмарного середовища (AWS S3).

Опис створення копії баз даних:

- створення клієнту для бази даних завдяки реалізованому плагіну;
- запит на створення бекапу до бази даних;
- створення копії.

Опис завантаження копії до хмарного середовища:

- підключення до файлового сховища – AWS S3;
- відкриття файлового дескриптору;
- використовуючи splice – експортування даних з локального середовища у віддалене;
- перевірка відправлених даних з використанням чексум;

- якщо всі чексуми рівні – дані успішно завантажені;
- інакше – сповіщення користувача у невідповідності чексум та копій.

Опис процесу оперування резервними копіями баз даних:

- отримання запиту;
- аналіз запиту на тип бази даних, таблиці, простору імен, сервера, де працює minion;
- перевірка на наявність плагіну запитуваної бази даних
- у разі відсутності плагіну – надсилання сповіщення користувачу та припинення поточного процесу;
- у разі наявності плагіну – створення бекапу (процес описаний вище)

завантаження бекапу до віддаленого сховища (процес описаний вище)

2.4 Аналіз безпеки даних

Будь-яка система повинна бути безпечною у використанні та запобігати виникнення проникнень.

Авторизація до веб-сервісу здійснюється завдяки авторизації через Basic auth або LDAP. Більше того, запити валідуються і тому SQL інєкції неможливі. Між користувачем та вебсервісом використовується HTTPs. Окрім цього валідується TGS тікети від Kerberos. Бекапи, які експротують до віддаленого сховища, передаються по безпечному каналу використовуючи TLS, який шифрується. Для знаходження DNS запису – використовується DNSSEC.

2.5 Висновки по розділу

У цьому розділі було проаналізовано багато аспектів даного проекту. Розроблювана система зазнала детального аналізу з усіх сторін технологічних рішень. Було змодельовано та побудовано систему, яка вважається відповідною до розроблюваного програмного продукту. Повний аналіз мов програмування показав переваги та недоліки написання на тій чи іншій мові програмування. Було обрано за основу мову Golang.

Було детально розглянуто алгоритм роботи кожного з сервісів даної системи. Побудовані діаграми, які наочно вказували на взаємодію користувача та обьектів. Окрім цього, було ще запроваджено діаграму, яка пояснює взаємодію процесів між сервісами.

Доведено використання RPC як основний механізм для спілкування між сервісами. Більше того, з HBase. Було вирішено проблему з веб-інтерфейсом. Веб-браузери працюють тільки з REST стилем. Тому було введено реалізацію трансформування REST запитів у RPC та навпаки.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Тестування – обов’язкова частина у розробці продукту. Розрізняють ручне та автоматизоване тестування.

У видадку автоматизованого тестування – процес розробки має власний механізм тестування У цьому випадку. Цей метод дозволяє швидко та без втручання інженера тестувати продукти.

Під час розробки системи, тестування відіграє дуже важливу роль. Розроблена система повинна бути протестовано як суцільно, так і розділено для верифікаї даних.

3.2 Опис процесів тестування

Кожна система повинна бути протестованою у разі її використання. Для деталізованого тестування було створено матрицю трасування, яка наглядно показує випадки помилок розробленого програмного забезпечення. Матрицю трасування задано у Таблиці 3.1.

Таблиця 3.1 - Матриця трасування функціональних вимог

Вимоги/ Випадки	Не працює клавіша	Сервіс не реагує на клавішу	Сервіс зреагував на клавішу, не було повідомлення	Не завантажено	Не авторизований для сервіс завантаження даних	Створена копія з неправильними параметрами	Успішне створення та завантаження копії у хмарне
Миттєве створення копії під час натискання клавіши	X	X	X	X	X	X	X
Створювати копію за заданими параметрами				X	X	X	X
Завантажити копію у хмарне сховище			X		X		X

Після аналізу матриці трасування з’ясовано, які випадки ще не покриті кодом та функціоналом для забезпечення повної надійності.

Так як реалізовано 3 основних сервіса для розроблюваної системи – dashboard, minion, master. Кожен з них повинен бути протестованим окремо від інших. Кожен виконує свою функцію та повинен виконувати її бездоганно. Так

як вже було протестовано за функціональними вимогами, потрібно проаналізувати сервіси за їх відповідність.

Dashboard. Сервіс відповідає за веб-застосунок, який відображає інформацію у веб-браузері. Для виведення конкретної інформації сервіс запитує master сервіс. Виникають наступні вимоги для такого сервісу:

- повернення HTML сторінки
- підтримка REST
- у разі помилки – виведення повідомлення
- вміти взаємодіяти з master сервісом

Master. Центральний сервіс, який зберігає поточну інформацію про minion-ів. Надає їм задачі. Контролю стан сервісів. Зберігає інформацію про створені резервні копії баз даних. Виникають наступні вимоги для такого сервісу:

- після перезагрузки сервісу – дані повинні залишитись
- збереження метаінформації про створенні копії
- у разі помилки одного з minion-ів, сервіс повинен змінити інформативний тєг цього сервісу на неактивний.

Minion. Сервіс-працівник, який встановлюється поряд з базами даних. Повинен реагувати на надані задачі від master сервісу. Вміти використовувати плагіни для взаємодії з базами даних. Вимоги:

- створення копій таблиць
- тригеритись мастером
- створювати заплановані копії
- віддавати статус master-у

Для тестування повинні бути виконані наступні умови. Підключення до інтернету. Задеплоїні коректно сервіси та запущені.

Загальні тест-кейси для розробленого веб-сервісу – dashboard.

Таблиця 3.2 – Загальній тест-кейси

Ідентифікатор	Опис
GC-1	Перевірити процес запуску сервісу
GC-2	Перевірити візаульно сторінку. Перевірка сдвигів.
GC-3	Перевірити, що клавіши баз даних натискаються і редиректять на сторінку з цією базою даних.
GC-4	Перевірити, що “Instant” натискається і сповіщює про вдалий запуск бекапу.
GC-5	Перевірити, що “Schedule” натискається і зявляється сповіщення про вдалий запуск бекапу.
GC-6	Перевірити, чи можливо задати запланований бекап без дати.
GC-7	Перевірити, що перехід на головну сторінку працює.
GC-8	Перевірити, що відображається сторінка з логування сервісів.
GC-9	Перевірити, що з’являються нові агенти у разі їх додавання
GC-10	Перевірити, що статус змінюється у разі неактовної чи активної участі агенту у healthcheck.
GC-11	Перевірити, що minion нотифікує master про свій стан.
GC-12	Перевірити, що minion здатний створювати бекапи

Продовження таблиці 3.2

GC-13	Перевірити, що master зберігає інформацію про Minion-ів.

Основна ідея будь-якого розроблюваного програмного продукту – його функціонування. Тому у разі забезпечення тестування його функціонування – вплине на розвиток та запит цього програмного продукту. Функціональні тест-кейси перелічено у таблиці 3.3.

Таблиця 3.3 – Функціональні тест-кейси

Ідентифікатор	Опис
FC-1	Перевірити, що minion створює резервні копії баз даних за заданим часом та миттєво
FC-2	Перевірити, щоб зберігалась універсальність інтерфейсу для взаємодії з базою даних
FC-3	Перевірити, що Master, Minion зберігають свій стан на локальній файловій системі.
FC-4	Перевірити, що Minion сервіс нотифікує Master за заданим інтервалом.
FC-5	Перевірити, що Dashboard трансформує запити з REST на RPC.

Звіт тестування програмного продукту для усіх тест-кейсів показано в таблиці 3.4.

Таблиця 3.4 – Звіт тестування

Ідентифікатор тесту	Очікуваний результат	Фактичний результат	Статус
GC-1	Сервіс запускається коректно.	Сервіс запускається коректно.	Пройдено
GC-2	Здвиги відсутні при генеруванні сторінки.	Здвиги відсутні при генеруванні сторінки.	Пройдено
GC-3	Клавіши з базами даних натискаються та переходять на наступну сторінку	Клавіши з базами даних натискаються та переходять на наступну сторінку.	Пройдено
GC-4	Натискається та створюється бекап при настиканні клавіши “Instant”	Натискається та створюється бекап при настиканні клавіши “Instant”	Пройдено
GC-5	Натискається та створюється бекап при настиканні клавіши “Schedule”	Натискається та створюється бекап при настиканні клавіши “Schedule”	Пройдено
GC-6	Задання бекапу без дати - неможливе	Задання бекапу без дати - неможливе	Пройдено

Продовження таблиці 3.4

GC-7	Перехід на головну сторінку - працює	Перехід на головну сторінку – працює	Пройдено
GC-8	Відображається сторінка з логуванням	Відображається сторінка з логуванням	Пройдено
GC-9	З’являються нові агенти у разі їх додавання	З’являються нові агенти у разі їх додавання	Пройдено
GC-10	Статус змінюється у разі неактовної чи активної участі агенту у healthcheck.	Статус змінюється у разі неактовної чи активної участі агенту у healthcheck.	Пройдено
GC-11	Minion нотифікує master про свій стан.	Minion нотифікує master про свій стан.	Пройдено
GC-12	Minion здатний створювати бекапи	Minion здатний створювати бекапи	Пройдено
GC-13	Master зберігає інформацію про Minion-ів.	Master зберігає інформацію про Minion-ів.	Пройдено
FC-1	Minion створює резервні копії баз даних за заданим часом та миттєво	Minion створює резервні копії баз даних за заданим часом та миттєво	Пройдено

Продовження таблиці 3.4

FC-2	Збегіється універсальність інтерфейсу для взаємодії з базою даних	Зберігається універсальність інтерфейсу для взаємодії з базою даних	Пройдено
FC-3	Master, Minion зберігають свій стан на локальній файловій системі	Master, Minion зберігають свій стан на локальній файловій системі	Пройдено
FC-4	Minion сервіс нотифікує Master за заданим інтервалом	Minion сервіс нотифікує Master за заданим інтервалом	Пройдено
FC-5	Dashboard трансформує запити з REST на RPC.	Dashboard трансформує запити з REST на RPC.	Пройдено

3.3 Опис контрольного прикладу

Розглянемо детально тест-кейс FC-1: Minion створює резервні копії баз даних за заданим часом та миттєво.

Таблиця 3.5 – Контрольний приклад

Мета тесту	Перевірка того, що minion створює бекапи за рокладом
Початковий стан	Відкрита сторінка веб-інтерфейсу

Продовження таблиці 3.5

Вхідні дані	- база даних – Hbase - простір імен – events - socket інформація агента
Схема проведення тесту	Введення вхідних даних. Перегляд результату у таргетовому середовищі.
Очікуваний результат	Створена резерная копія, яка вже завантажена у хмарне сховище.
Стан програмного продукту після проведення випробувань	Додано записи: - останній час коли було створено копію - час за який було встановлено регулярні бекапи.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Дана система – кросплатформенна. Дозволяє без додаткових інструментів встановити на сервер та запустити звичаний бінарний файл.

Достатньо скомпілювати виконавши «GOOS=linux go build cmd/<service name>» та встановити на сервер запустивши цей бінарний файл. Для конфігурації сервісів потрібно використати готові конфігураційні файли.

4.2 Робота з програмним забезпеченням

Розроблений програмний продукт передбачає наступні дії:

1. Відкрити головну сторінку
2. Натиснути на одну з баз даних з існуючих minion-ів.
3. Обрати одну з таблиць
4. Створити миттєву копію таблицю та завантажити її до хмарного середовища або запланувати створення копії.

ВИСНОВКИ

Під час виконання даної бакалаврської роботи було проаналізовано проблеми централізованих систем створення резерних копій баз даних та їх відсутність на ринку. Проаналізовано відомі технічні рішення створення резервних копій, їх переваги та недоліки.

Було змодельовано та розроблено автоматизовано систему, яка здатна контролювати резервні копії баз даних. При цьому можна через веб-застосунок планувати створення копій або створювати миттєві копії. Було проаналізовані мови програмування для реалізації такої системи. Зважаючи на сутність запитів в такій системі було порівняно RPC і REST та обрано RPC як основний протокол для спілкування між сервісами.

Було протестовано сервіси відповідно до функціональних вимог.

Розроблено документацію, схеми процесів сервісів, варіанти використання та керівництво для користувача.

ПЕРЕЛІК ПОСИЛАНЬ

1) Top 5 cloud-computing vendors [Text] // Forbes. — 2017. — Access mode: https://www.forbes.com/sites/bobevans1/2017/11/07/ the-top-5-cloud-computing-vendors-1-microsoft-2-amazon-3-ibm-4-salesforce-5-sap/.
2) McGehee, Shawn. SQL Server backup and Restore [Text] / Shawn McGehee. — [S. l.] : Simple Talk, 2012. — P. 50–78.
3) Top 30 most popular database management software [Text] // Journal Software Testing. — 2019. — https://www.softwaretestinghelp.com/databasemanagement-software/.
4) NoSQL Databases: The definitive Guide [Text] // Blog pandorafms. — 2018. — Access mode: https://blog.pandorafms.org/ nosql-databases-the-definitive-guide/.
5) Wilder, Bill. Cloud architecture patterns [Text] / Bill Wilder. — [S. l.] : O'Reilly, 2012. — P. 10–30.
6) Greengard, Samuel. The internet of Things [Text] / Samuel Greengard. — [S. l.] : MIT Press, 2015. — P. 80–111.
7) Sbook, Donald Miner & Adam. MapReduce Design Patterns [Text] / Donald Miner & Adam Sbook. — [S. l.] : OREILLY, 2012. — P. 54–120. ´
8) Traditional ETL vs ELT on Hadoop [Text] // Bitwise. — 2017. — https://www.bitwiseglobal.com/blogs/traditional-etl-vs-elt-on-hadoop/.
9) Simon, Phil. Too big to ignore [Text] / Phil Simon. — [S. l.] : Wiley, 2015. — P. 1–50.
10) Gunarathne, Thilina. Hadoop Cookbook / Thilina Gunarathne. — 2015. — Hadoop Cookbook.
11) Rajan, Claire. Database Administrator Backup/Recovery and Network Administration [Text] / Claire Rajan // Database Administrator. — [S. l. : s. n.], 2018. — P. 53–526.
12) SQLShack [Text] // SQLShack. — 2019. — https://www.sqlshack.com/sql-server-database-backup-tools.

13) Iperius [Text] // Iperius. — 2019. — https://www.iperiusbackup.com/backup-database.aspx.

14) Urbackup [Text] // Urbackup. — 2019. — http://www.urbackup.org.

15) AWSBackup [Text] // AWS. — 2019. — https://console.aws.amazon.com/backup/home?region=us-east-1.

16) HandyBackup [Text] // HandyBackup. — 2019. — https://www.handybackup.net/database-backup.shtml.

17) QualityNOC [Text] // QualityNOC. — 2019. — http://www.qualitynoc.com/5-open-source-backup-software-for-linux.

18) List of C family languages [Text] // Wikipedia. — 2019. — Access mode: https://en.wikipedia.org/wiki/List_of_C-family_programming_languages.

19) Bailey, Tim / Tim Bailey // C programming language. — [S. l. : s. n.], 2005. — P. 1–65, 79–84.

20) Bailey, Tim / Tim Bailey // C programming language. — [S. l. : s. n.], 2005. — P. 115–123.

21) Stevanovic, Milan. C and C++ Compiling [Text] / Milan Stevanovic. — [S. l.] : APRESS, 2014.

22) When to use dynamic linking and static linking [Text] // IBM. — 2019. — Access mode: https://www.ibm.com/support/knowledgecenter/en/ssw_aix_71/com.ibm.aix.performance/when_dyn_linking_static_linking.htm.

23) About linkers and dynamic/static linking [Text] : Rep. / Indiana University ; Executor: Indiana University : 2019.

24) Bailey, Tim / Tim Bailey // C programming language. — [S. l. : s. n.], 2005. — P. 68–79.

25) C API libhdfs [Text] // Apache Hadoop. — 2018. — Access mode: https: //hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/LibHdfs.html.